**과목명: 시스템프로그래밍**

**1 분반**

**<<Project #1>>**

**서강대학교 [컴퓨터 공학과]**

**[20171653]**

**[신나현]**

목 차

1. **프로그램 개요**
2. **프로그램 설명**
   1. 프로그램 흐름도
3. **모듈 정의**
   1. main()
      1. 사용 변수
      2. 모듈 기능
   2. Get\_Command()
      1. 사용 변수
      2. 모듈 기능
   3. Read\_Opcode()
      1. 사용 변수
      2. 모듈 기능
   4. Hash\_Function(char \*arr)
      1. 사용 변수
      2. 모듈 기능
   5. Print\_All\_Command()
      1. 사용 변수
      2. 모듈 기능
   6. Current\_Directory\_File()
      1. 사용 변수
      2. 모듈 기능
   7. View\_History()
      1. 사용 변수
      2. 모듈 기능
   8. Quit\_Sicsim()
      1. 사용 변수
      2. 모듈 기능
   9. Push\_Command(char \*arr)
      1. 사용 변수
      2. 모듈 기능
   10. dump(int start, int end)
       1. 사용변수
       2. 모듈 기능
   11. Edit(int addr, int value)
       1. 사용변수
       2. 모듈 기능
   12. Fill(int start, int end, int value)
       1. 사용변수
       2. 모듈 기능
   13. Reset()
       1. 사용 변수
       2. 모듈 기능
   14. Print\_Opcode(char \*arr)
       1. 사용 변수
       2. 모듈 기능
   15. Opcode\_List()
       1. 사용 변수
       2. 모듈 기능
4. **전역 변수 정의**
   1. unsigned char memory[16][65536]
   2. history \*head
   3. hash \*hashhead[20]
   4. int dumpend
   5. int dumpstart
5. **코드 설명**
   1. 20171653.h
   2. 20171653.c
6. **프로그램 개요**

이 프로그램은 앞으로 구현하게 될 SIC/XE 머신을 구현하기 위한 전 단계로서 어셈블러, 링크, 로더들을 실행하게 될 셸(shell)과 컴파일을 통해서 만들어진 object 코드가 적재 되고 실행될 메모리공간과 mnemonic (ADD, COMP, FLOAT, etc ...)을 opcode 값으로 변환하 는 OPCODE 테이블과 관련 명령어들을 구현하는 프로그램입니다.

1. **프로그램 설명**
   1. 프로그램 흐름도

화이트보드, 텍스트이(가) 표시된 사진

자동 생성된 설명

1. **모듈 정의**
   1. **main()**
      1. 사용 변수

없음.

* + 1. 모듈 기능

Read\_Opcode() 함수를 호출하여 opcode.txt파일을 읽어와 해쉬 테이블에 저장하게한다. while loop를 이용해 무한 루프에서 Get\_Command()함수를 호출한다.

* 1. **Get\_Command()**
     1. 사용 변수

char str[100]: 사용자로부터 명령어를 입력 받을 때 사용하여 입력된 내용 저장.

char save[100]: Push\_Command함수를 이용해 입력 받았던 명령어를 그대로 history에 저장하기 위해 그 내용을 저장.

char use[100]: Print\_Opcode함수에 념겨줘 사용하기위해 save[100]의 내용 strcpy 이용해 복사해 둠.

char command[100]: 제일 첫 단어를 저장해 각 명령어에 맞는 함수를 실행시키기 위해 사용, 유효한 명령어인지 확인하기 위해 사용.

int num[5]: 첫 단어 이후에 16진수가 입력되었을 때 그 숫자들을 저장하는데 사용.

int i: for문에 사용.

int comma: 입력된 명령어에 ‘,’가 몇 개 인지 확인.

int len; 입력된 명령어의 길이 저장.

char \*temp: strtok로 string을 자를 때 사용.

int cnt: 첫 단어 뒤에 들어오는 숫자의 수를 세서 유효한 명령어인지 판단하는데 사용.

* + 1. 모듈 기능

사용자에게서 명령어를 입력 받아 유효한 명령어인 경우 그 명령어에 맞는 함수를 호출하여 그 기능을 수행하도록 하고 Puch\_Command(char\*) 함수를 호출하여 히스토리에 저장해준다. 유효하지 않은 명령어인 경우 에러 메세지를 출력한다.

* 1. **Read\_Opcode()**
     1. 사용 변수

int n: 몇 번째 해쉬에 명령어를 저장할지 나타냄.

int opcode: opcode.txt에서 읽어온 16진수인 opcode를 저장.

char mnemonic[100]: opcode.txt에서 읽어온 instruction name을 저장.

char num[100]: opcode.txt에서 읽어온 format을 저장.

hash \*new: 링크드리스트로 해쉬테이블을 구현하기위해 사용.

hash \*temp: 링크드리스트로 해쉬테이블을 구현하기위해 사용.

FILE \*ofp: opcode.txt를 여는데 사용.

* + 1. 모듈 기능

opcode.txt 파일을 열어 그 내용을 while문을 이용해 한 줄씩 읽어와서 Hash\_Function(char\*)을 호출하여 return받은 번째의 해쉬에 링크드리스트의 형태로 opcode를 저장한다.

* 1. **Hash\_Function(char \*arr)**
     1. 사용 변수

sum: 몇 번째 hash에 저장할지 나타내기위해 각 글자의 아스키코드값을 더한 값을 저장.

* + 1. 모듈 기능

String에 대한 해쉬값을 반환해준다.

* 1. **Print\_All\_Command()**
     1. 사용 변수

없음.

* + 1. 모듈 기능

Shell에서 실행가능한 모든 명령어들의 리스트를 화면에 출력해준다.

* 1. **Current\_Directory\_File**()
     1. 사용 변수

DIR \*directory: 디렉토리를 열어주는데 사용.

struct dirent \*di: directory에 저장된 디렉토리의 정보를 가지고 파일이나 폴더를 받아 옴.

struct stat statbuf: 파일인지 폴더인지 알려줌.

* + 1. 모듈 기능

현재 디렉토리에 있는 파일들을 출력해주며 파일 이름 옆에는 ‘\*’ 표시를, 디렉토리는 ‘/’ 표시를 해줌.

* 1. **View\_History()**
     1. 사용 변수

int num: 몇번재로 히스토리에 저장된 명령어인지 나타냄.

history \*temp: 링크드리스트를 head부터 따라가며 히스토리를 출력하기 위해 사용.

* + 1. 모듈 기능

지금까지 사용자에게서 입력받아 실행된 유효한 명령어들을 순서대로 번호와 함께 보여준다.

* 1. **Quit\_Sicsim()**
     1. 사용 변수

history \*del: 링크드리스트 삭제를 위해 사용.

hash \*hashdel: 링크드리스트 삭제를 위해 사용.

* + 1. 모듈 기능

링크드리스트를 모두 free해준 뒤 sicsim을 종료해준다.

* 1. **Push\_Command(char\* arr)**
     1. 사용 변수

history \*save: 링크드리스트를 생성하고 삽입하는데 사용.

history \*temp: 링크드리스트를 탐색하는데 사용.

* + 1. 모듈 기능

입력 받은 유효한 명령어를 history에 링크드리스트 형태도 저장해준다.

* 1. **dump(int start, int end)**
     1. 사용변수

int startaddr: start번지가 memory의 몇번째 줄인지 나타냄.

int endaddr: end번지가 memory의 몇번째 줄인지 나타냄.

* + 1. 모듈 기능

start번지부터 end번지까지의 memory내용을 출력해준다.

* 1. **Edit(int addr, int value)**
     1. 사용변수

없음.

* + 1. 모듈 기능

addr번지의 값을 value에 지정된 값으로 변경해준다.

* 1. **Fill(int start, int end, int value)**
     1. 사용변수

없음.

* + 1. 모듈 기능

start번지부터 end번지까지의 값을 Edit함수를 호출하여 value에 지정된 값으로 변경해준다.

* 1. **Reset()**
     1. 사용 변수

없음.

* + 1. 모듈 기능

Edit함수를 호출하여 메모리 전체를 전부 0으로 변경시킨다.

* 1. **Print\_Opcode(char\*)**
     1. 사용 변수

char mnemonic[100]: opcode를 출력할 명령어 저장.

char \*token: strtok를 이용해 문자열을 자르는데 이용.

int num: 명령어에 해당하는 hash값을 저장.

hash \*temp: 해쉬테이블을 탐색하여 opcode를 찾는데 이용

* + 1. 모듈 기능

유효한 mnemonic인 경우 명령어에 해당하는 opcode를 출력해주고 1을 반환해주고 유효하지 않은 mnemonic인 경우 0을 반환해준다.

* 1. **Opcode\_List()**
     1. 사용 변수

hash \*temp: 링크드리스트로 연결된 해쉬 테이블을 탐색하는데 이용.

* + 1. 모듈 기능

해쉬테이블의 내용을 출력해준다.

1. **전역 변수 정의**
   1. **unsigned char memory[16][65536]**

1Mbyte의 가상 메모리 공간.

* 1. **history \*head**

history를 링크드리스트 형태로 저장하기 위해 사용.

* 1. **hash \*hashhead[20]**

opcode hash table을 링크드리스트 형태로 저장하기 위해 사용

* 1. **int dumpend**

dump를 실행할 때 end 주소를 저장하는 전역 변수.

* 1. **int dumpstart**

dump를 실행할 때 end 주소를 저장하는 전역 변수.

1. **코드 설명**
   1. **20171653.h**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<dirent.h>

#include<sys/stat.h>

typedef struct HashTable{

int opcode;

char command[100];

char format[100];

struct HashTable \*next;

}hash;

typedef struct History{

char command[100];

struct History \*next;

}history;

void Get\_Command(); //명령어 입력받는 함수

void Read\_Opcode(); //opcode.txt읽어오는 함수

int Hash\_Function(char \*arr); //hash값 생성

int Print\_Opcode(char \*arr); //mnemonic에 해당하는 opcode 출력

void Opcode\_List(); //opcode hash table 출력

void Print\_All\_Command(); //h[elp]

void Current\_Directory\_File(); //d[ir]

void Quit\_Sicsim(); //q[uit]

void View\_History(); //hi[story]

void Push\_Command(char \*arr); //history를 링크드리스트 형태로 저장

void dump(int start, int end); //du[mp]

void Edit(int addr, int value); //e[dit]

void Fill(int start, int end, int value); //f[ill]

void Reset(); //reset

* 1. **20171653.c**

#include "20171653.h"

unsigned char memory[16][65536];

history \*head;

hash \*hashhead[20];

int dumpend=-1;

int dumpstart;

/\*

main함수\*/

int main(void){

Read\_Opcode();

while(1){

Get\_Command();

}

}

/\*

Print\_All\_Command()

h[elp]

shell에서 실행가능한 모든 명령어 리스트 출력

\*/

void Print\_All\_Command(){

printf("h[elp]\n");

printf("d[ir]\n");

printf("q[uit]\n");

printf("hi[story]\n");

printf("du[mp] [start, end]\n");

printf("e[dit] address, value\n");

printf("f[ill] start, end, value\n");

printf("reset\n");

printf("opcode mnemonic\n");

printf("opcodelist\n");

}

/\*

Current\_Directory\_File()

d[ir]

현재 디렉터리의 파일 출력

\*/

void Current\_Directory\_File(){

DIR \*directory=opendir("./");

struct dirent \*di;

struct stat statbuf;

if(directory !=NULL){

while((di = readdir(directory))!=NULL){

lstat(di->d\_name,&statbuf);

if(S\_ISREG(statbuf.st\_mode))

printf("%s\*\t", di->d\_name);

else if(S\_ISDIR(statbuf.st\_mode)&&S\_IXUSR)

printf("%s/\t", di->d\_name);

}

printf("\n");

}

closedir(directory);

}

/\*

Quit\_Sicsim()

q[uit]

링크드리스트로 구현된 history와 opcode hash table을 모두 free하여 삭제해준 뒤 sicsim 종료

\*/

void Quit\_Sicsim(){

history \*del;

hash \*hashdel;

while(head != NULL){

del = head;

head = head->next;

free(del);

}

for(int i=0; i<20; i++){

while(hashhead[i]!=NULL){

hashdel = hashhead[i];

hashhead[i] = hashhead[i]->next;

free(hashdel);

}

}

exit(0);

}

/\*

View\_History()

hi[story]

링크드르스트로 구현된 history를 head부터 따라가면서 사용한 명령어 순서대로 번호와 함께 보여줌

\*/

void View\_History(){

int num=1;

history \*temp = head;

while(temp!=NULL){

printf("%d %s\n", num, temp->command);

temp = temp->next;

num++;

}

}

/\*

Push\_Command(char \*arr)

command를 history의 링크드리스트의 맨 뒤에 삽입함.

\*/

void Push\_Command(char \*arr){

history \*save = NULL;

history \*temp;

save = (history\*)malloc(sizeof(history));

strcpy(save->command, arr);

save->next=NULL;

if(head == NULL){

head = save;

}

else{

temp = head;

while(temp->next != NULL){

temp = temp->next;

}

temp->next = save;

}

}

/\*

dump(int start, int end)

du[mp] start, end

start번지부터 end번지까지의 메모리의 내용을 <출력하는 메모리의 주소를 5자리 16진수 형태로, 메모리의 내용을 16진수 형태로, 메모리의 내용에 대응하는 ASCII code형태>로 출력해줌

start번지와 end번지가 있는 줄의 각각 첫번지~start-1번지와 end+1번지~끝번지는 16진수로 출력해주지않고 ASCII코드부분역시 '.' 으로 출력해주므로 case를 나눠 출력해줌

\*/

void dump(int start, int end){

int startaddr=start/16;

int endaddr=end/16;

for(int i= startaddr; i<=endaddr; i++){

printf("%05X ", i\*16);

if(startaddr == endaddr){ //start와 end 번지가 한줄에 있는경우

for(int j=0; j<start%16; j++){

printf(" ");

}

for(int j=start%16; j<=end%16; j++){

printf("%02X ", memory[j][i]);

}

for(int j=end%16+1; j<16; j++){

printf(" ");

}

printf("; ");

for(int j=0; j<start%16; j++){

printf(".");

}

for(int j=start%16; j<=end%16; j++){

if(memory[j][i]>=0x20 && memory[j][i]<=0x7E){

printf("%c",memory[j][i]);

}

else{

printf(".");

}

}

for(int j=end%16+1; j<16; j++){

printf(".");

}

}

else if(i== startaddr){ //start번지가 있는 줄

for(int j=0; j<start%16; j++){

printf(" ");

}

for(int j=start%16; j<16; j++){

printf("%02X ", memory[j][i]);

}

printf("; ");

for(int j=0; j<start%16; j++){

printf(".");

}

for(int j=start%16; j<16; j++){

if(memory[j][i]>=0x20 && memory[j][i]<=0x7E){

printf("%c",memory[j][i]);

}

else{

printf(".");

}

}

}

else if(i== endaddr){ //end번지가 있는 줄

for(int j=0; j<=end%16; j++){

printf("%02X ", memory[j][i]);

}

for(int j=end%16+1; j<16; j++){

printf(" ");

}

printf("; ");

for(int j=0; j<=end%16; j++){

if(memory[j][i]>=0x20 && memory[j][i]<=0x7E){

printf("%c",memory[j][i]);

}

else{

printf(".");

}

}

for(int j=end%16+1; j<16; j++){

printf(".");

}

}

else{ //중간

for(int j=0; j<16; j++){

printf("%02X ", memory[j][i]);

}

printf("; ");

for(int j=0; j<16; j++){

if(memory[j][i]>=0x20 && memory[j][i]<=0x7E){

printf("%c",memory[j][i]);

}

else{

printf(".");

}

}

}

printf("\n");

}

}

/\*

Edit(int addr, int value)

e[dit]

메모리에서 해당 address의 값을 value로 바꿔 줌\*/

void Edit(int addr, int value){

memory[addr%16][addr/16]=value;

}

/\*

Fill(int start, int end, int value)

f[ill]

for문에서 Edit함수를 호출하여 start에서 end번지까지 value로 바꿔 줌\*/

void Fill(int start, int end, int value){

for(int i=start; i<=end; i++){

Edit(i, value);

}

}

/\*

Reset()

reset

for문에서 Edit함수를 호출하여 메모리를 전부 0으로 바꿔 줌 \*/

void Reset(){

for(int i=0; i<=0xFFFFF; i++){

Edit(i, 0);

}

}

/\*

Read\_Opcode()

opcode.txt 읽어오는 함수

fopen으로 opcode.txt를 열어와서 파일의 끝까지 읽어온다.

읽어온 내용을 opcode hash table에 링크드리스트 형태로 저장해준다.

\*/

void Read\_Opcode(){

int n;

int opcode;

char mnemonic[100];

char num[100];

hash \*new = NULL;

hash \*temp;

FILE \*ofp = fopen("opcode.txt", "r");

if(ofp == NULL){

printf("opcode.txt ERROR!"); //opcode파일이 없으면 ERROR출력

return;

}

for(int i=0; i<20; i++){

hashhead[i] = NULL;

}

while(fscanf(ofp, "%X%s%s", &opcode, mnemonic, num)!=EOF){

new = (hash\*)malloc(sizeof(hash));

new->opcode = opcode;

strcpy(new->command, mnemonic);

strcpy(new->format, num);

n=Hash\_Function(mnemonic); //저장할 hash값을 return받아온다.

if(hashhead[n] == NULL){

hashhead[n] = new;

}

else{

temp = hashhead[n];

while(temp->next != NULL){

temp = temp->next;

}

temp->next = new;

}

//printf("%X %s %s\n", new->opcode, new->command, new->number);

}

fclose(ofp); //파일을 닫음

}

/\*

Hash\_Function(char \*arr)

hash 값 결정해서 return해줌

\*/

int Hash\_Function(char \*arr){

int sum=0;

for(int i=0; i<strlen(arr); i++){

sum+=(int)arr[i];

}

return sum%20;

}

/\*

Print\_Opcode(char \*arr)

opcode mnemonic

명령어에 해당하는 opcode를 출력하기 위해 링크드리스트로 구현된 hash table을 탐색한다

유효한 mnemonic인 경우 그에 해당하는 opcode를 출력하고 1을 return하고

유효하지않은 mnemonic인 경우 0을 return 한다

\*/

int Print\_Opcode(char \*arr){

char mnemonic[100];

char \*token;

int num;

hash \*temp;

token = strtok(arr, " ");

token = strtok(NULL," ");

strcpy(mnemonic, token);

num = Hash\_Function(mnemonic);

temp = hashhead[num];

while(strcmp(temp->command, mnemonic) != 0){

temp = temp->next;

if(temp == NULL){

return 0;

}

}

printf("opcode is %X\n", temp->opcode);

return 1;

}

/\*

Opcode\_List()

opcodelist

링크드리스트로 구현된 opcode hash table을 차례로 탐색하면서 그 내용 출력해준다\*/

void Opcode\_List(){

hash \*temp;

for(int i=0; i<20; i++){

printf("%d : ", i);

temp = hashhead[i];

if(temp != NULL){

while(temp->next!=NULL){

printf("[%s,%X] -> ", temp->command, temp->opcode);

temp = temp->next;

}

printf("[%s,%X]", temp->command, temp->opcode);

}

printf("\n");

}

}

/\*

Get\_Command()

command를 입력받아 유효한 명령어인 경우 각각의 명령어에 맞는 함수를 실행해 주고

유효하지 않은 경우엔 ERROR를 출력해 준다\*/

void Get\_Command(){

char str[100]={0};

char save[100]={0};

char use[100]={0};

char command[100] = {0};

int num[5]={-1, -1, -1,-1,-1};

int i, comma=0;

int len;

char \*temp=NULL;

int cnt=0;

printf("sicsim> ");

fgets(str, sizeof(str), stdin); //명령어 입력받음

len = strlen(str);

str[len-1] = '\0';

for(i=0; i<len;i++){

if(str[i] == ','){

comma++;

}

}//dump [start,end], edit, fill 인경우엔 0이 아님

strcpy(save, str); //history에 push하기위해 원래 명령 그대로 저장

temp = strtok(str, ", ");

strcpy(command, temp); //명령어의 첫단어만 잘라서 저장

//printf("%s", command);

temp = strtok(NULL, ", ");

while(temp != NULL){

sscanf(temp, "%X", &num[cnt]); //숫자부분 저장

cnt++;

temp = strtok(NULL,", ");

}

if(strcmp(command,"h")==0 || strcmp(command, "help")==0){

if(cnt!=0)

printf("command ERROR\n");

else{

Push\_Command(save);

Print\_All\_Command();

}

}// h[elp]

else if(strcmp(command, "q")==0 || strcmp(command,"quit")==0){

if(cnt!=0)

printf("command ERROR!\n");

else{

Push\_Command(save);

Quit\_Sicsim();

}

}//q[uit]

else if(strcmp(command, "hi")==0 || strcmp(command, "history")==0){

if(cnt!=0)

printf("command ERROR!\n");

else{

Push\_Command(save);

View\_History();

}

}//hi[story]

else if(strcmp(command, "d")==0 || strcmp(command, "dir")==0){

if(cnt!=0){

printf("command ERROR!\n");

}

else{

Push\_Command(save);

Current\_Directory\_File();

}

}//d[ir]

else if(strcmp(command, "opcode")==0 && cnt==1){

//Push\_Command(save);

strcpy(use, save);

int sw = Print\_Opcode(use);

if(sw == 1){

Push\_Command(save); //opcode.txt에 존재하는 mnemonic일때만 history에 삽입

}

else{

printf("command ERROR!\n"); //존재하지않는 mnemonic인 경우 ERROR 출력

}

}//opcode mnemonic

else if(strcmp(command, "opcodelist")==0 && cnt==0){

Push\_Command(save);

Opcode\_List();

}//opcodelist

else if(strcmp(command, "dump")==0 || strcmp(command, "du")==0){

if(cnt>2){

printf("command ERROR!\n");

}

else{

if(comma>=2){

printf("command ERROR!\n");

}

else if(comma==1 && cnt==2){ //dump start, end

if(num[0]!=-1 && num[1]!=-1){

if((num[0]<=num[1]) && (num[0] >=0) &&(num[1]>=0) && (num[0]<=0xFFFFF) && (num[1]<=0xFFFFF)){

Push\_Command(save);

dump(num[0], num[1]);

dumpend = num[1];

dumpstart = num[0];

}

else{

printf("command ERROR!\n");

}

}

else{

printf("command ERROR!\n");

}

}

else if(comma==0){

if(num[0]==-1 && cnt==0){ //dump

Push\_Command(save);

dumpstart = dumpend+ 1;

if(dumpstart > 0xFFFFF){

dumpstart=0;

}

dumpend = dumpstart+159;

if(dumpend > 0xFFFFF){

dumpend = 0xFFFFF;

}

dump(dumpstart, dumpend);

}

else if(num[0]!=-1 && cnt==1){ //dump start

if(num[0]>=0 && num[0]<=0xFFFFF){

Push\_Command(save);

dumpstart = num[0];

dumpend = dumpstart+159;

if(dumpend>0xFFFFF){

dumpend=0xFFFFF;

}

dump(dumpstart, dumpend);

}

else{

printf("command ERROR!\n");

}

}

else{

printf("command ERROR!\n");

}

}

}

}//du[mp]

else if(strcmp(command, "e")==0 || strcmp(command, "edit")==0){

if(comma==1 && cnt==2){

if(num[0]>=0 && num[0]<=0xFFFFF && num[1]>=0 && num[1]<=0xFF){

Push\_Command(save);

Edit(num[0], num[1]);

}

else{

printf("command ERROR!\n");

}

}

else{

printf("command ERROR!\n");

}

}//e[dit]

else if(strcmp(command,"f")==0 || strcmp(command, "fill")==0){

if(comma == 2 && cnt==3){

if(num[0]>=0 && num[0]<=0xFFFFF && num[1]>=0 && num[1]<=0xFFFFF && num[0]<=num[1] && num[2]>=0 && num[2]<=0xFF){

Push\_Command(save);

Fill(num[0], num[1], num[2]);

}

else{

printf("command ERROR!\n");

}

}

else{

printf("command ERROR!\n");

}

}//f[ill]

else if(strcmp(command, "reset")==0 && cnt==0){

Push\_Command(save);

Reset();

}//reset

else{

printf("command ERROR!\n");

}

}